EXHIBIT I

Compatibility goals in the Apache Harmony Class Library

The following guidelines are currently PROPOSED and being discussed on the development mailing list `dev@harmony.apache.org`. Please direct comments and questions there.

The Harmony project class library ("**classlib**") effort is committed to producing a set of class library code that not only Java compliant but is also compatible with existing Java implementations. This page describes the class library code guidelines for ensuring that compatibility.

The "Reference Implementation"

The Harmony project 'classlib' effort is producing a set of class library code that is compatible with the Java SE 5.0 API specification.

At times (discussed below) it is necessary to augment the specification based on the behavior of the reference implementation (RI) of the Java SE 5.0 specification. In such cases we use the latest official release of the Java SE 5.0 RI.

We are aware that there are other compliant implementations of Java 5.0 available but there is only one reference implementation of the Java specification.

General Compatibility Guidelines

The following general guidelines apply when developing class library code for Apache Harmony:

Comply with the Java Specification

In the first instance we follow the description of each part of the class library given in the Java specification. To the large part, this adequately describes the behavior of the class library implementation including methods, exceptions, and serialization. However, where the specification is silent on a particular issue facing the implementor, or the described behavior is clearly illogical, then we follow the behavior of the reference implementation.

Follow the Reference Implementation

The Reference Implementation (RI) is used to resolve issues that are not adequately addressed in the specification. In such cases we use the RI as a guide to the compliant and expected behavior; however, developers *must* ensure that the behaviour of the RI is determined soley through exercising the public Java APIs -- specifically we avoid any notion of reverse engineering or exposition of the RI's implementation by exercising non-API types and methods.
There are a few occasions where both the specification is quiet on a given issue, and the RI exhibits behaviour that we would consider illogical. In such cases we discuss the issue on the Harmony developers' mailing list, and code the class libraries to do what the development community conclude is "the logical thing".

Do "the Logical Thing"

The final decision about how a piece of code should behave, on those rare occasions where the specification and RI do not provide a satisfactory answer, is reached by concensus on the Harmony developers' mailing list. Each issue is debated based on it's individual circumstances, but the developers are aware that breaking popular applications is invariably not "the logical thing" to do.

Once a decision has been made it is documented in the code comments and an issue raised in the Harmony JIRA issue tracking system to record the conclusion. It should be noted that very few issues need to be resolved this way.

Exception-throwing compatibility

There are a number of methods in the Java specification that describe the conditions under which exceptions are thrown. However, in most cases the specification does not describe all possible exceptions that may be thrown, the order of exception throwing (i.e. where there are multiple conditions that would result in an exception), and so on.

The Harmony class libary code aims to be fully compatible with the Reference Implementation (RI) of the Java Specification by matching the exception characteristics of each method.

Harmony **classlib** developers write test cases that deliberately cause exceptions to be thrown so that we can match exception behaviour like-for-like. Harmony class library code throws exceptions of the same runtime class (or a subtype of that runtime class) as the RI, other than in cases where the RI throws non-public types whereupon Harmony will throw an exception with the same public supertype.

Generally, we could refer to the following steps.

If RI is compliant with the Java Specification

We shall follow RI's behavior, that is, throws exactly the same exception or a subclass of the exception.

- But there are some cases that RI throws an implementation specific exception, which is not in the Java Specification, we shall throw an equivalent Harmony specific exception, or its superclass in the Java Specification.

Example

If RI throws `sun.io.MalformedInputException`, we can throw `org.apache.harmony.io.MalformedInputException` or `java.io.CharConversionException`.

If RI is not compliant with the Java Specification

- If the Java Specification describes the exceptional situation explicitly, and RI throws different kind of exception or even does not throw any exception, we shall discuss them case by case in harmony-dev mailing list.
  - If we decide to follow RI, we will raise an "Non-bug differences from Spec" JIRA.
  - If we decide to follow the Java Specification, we will raise an "Non-bug differences from RI" JIRA.
- If the Java Specification does NOT describe the exceptional situation explicitly, and RI's behavior is either hard to reproduce or illogical, we shall discuss them case by case. And we may decide to:
  - Follow RI
  - Follow the Java Specification
  - Implement the functions in our own way

Serialization compatibility

The Harmony class library code aims to be serialization compatible with the reference implementation (RI).

The Java Specification describes the serialized form of many Java types. Where given, **classlib** will follow the specified serialized form. When the serialized form is NOT given we will ensure that the serialization data is compatible with the RI by ensuring that objects serialized from the RI can be read by Harmony's **classlib** code, and vice versa.

Serialization tests are part of our regular test suite and typically rely on having the persistent form of serialized objects written from the RI stored alongside our test suite code.

To indicate that we are serialization-compatible we define an explicit Stream Unique Identifier (SUID) in each of the Harmony **classlib** serializable types that is equal to the SUID of the corresponding type in the RI.

Where the RI produces a serialized form that cannot be replicated by Harmony (e.g. the RI serialized form includes types that are not part of the Java specification) then Harmony cannot be serialization compatible with the RI, and will both make a prominent note of such in the relevant type's JavaDoc comment, and raise a JIRA issue that describes the incompatibility.
